Parallel Processing Letters

Assessing the performance of energy-aware mappings

Anne Benoit
Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
Anne.Benoit@ens-lyon.fr

and

Rami Melhem
Department of Computer Science, University of Pittsburgh
6429 Sennot Square, Pittsburgh, USA
melhem@cs.pitt.edu

and

Paul Renaud-Goud
Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
Paul.Renaud-Goud@ens-lyon.fr

and

Yves Robert
Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
and University of Tennessee Knoxville, TN, USA
Yves.Robert@ens-lyon.fr

Received December 2012
Revised March 2013
Communicated by Jack Dongarra and Bernard Tourancheau

Abstract

We aim at mapping streaming applications that can be modeled by a series-parallel graph onto a 2-dimensional tiled chip multiprocessor (CMP) architecture. The objective of the mapping is to minimize the energy consumption, using dynamic voltage and frequency scaling (DVFS) techniques, while maintaining a given level of performance, reflected by the rate of processing the data streams. This mapping problem turns out to be NP-hard, and several heuristics are proposed. We assess their performance through comprehensive simulations using the *StreamIt* workflow suite and randomly generated series-parallel graphs, and various CMP grid sizes.

Keywords: multicore, energy, period, optimization, DVFS, streaming applications.

1. Introduction

The *energy consumption* of computational platforms has recently become a critical problem, both for economic and environmental reasons [7]. To reduce energy consumption, processors can run at different speeds. Faster speeds allow for a faster execution, but they also lead to a much higher (superlinear) power consumption. Energy-aware scheduling aims at minimizing the energy consumed during the execution of the target application, both for computations and for communications. But the price to pay for a lower energy consumption usually is a much larger execution time, so this approach makes sense only if coupled with some performance bound to be achieved. In other words, we have a bi-criteria optimization problem, with one objective being energy minimization, and the other being performance-related.

In this paper, we aim at minimizing the energy consumption of streaming applications whose task graph is a series-parallel graph (SPG). Streaming applications, or workflows, are ubiquitous in many domains, as for instance image processing applications, astrophysics, meteorology, neuroscience, and so on [5, 12]. Most of these applications have simple and regular task graphs, such as linear chains, trees, fork-join graphs, or general SPGs (see Section 2 for a formal definition of SPGs). For instance, all the benchmarks of the *StreamIt* suite [13] are SPGs. The performance-related objective coupled with energy minimization is the *period* of the streaming application. Typically, a series of data sets enter the input stage and progress from stage to stage, following the dependencies of the application, until the final result is computed. Each stage has its own communication and computation requirements: it reads inputs from the previous stage(s), processes the data and outputs results to the next stage(s). The pipeline operates in a dataflow mode: after a transient behavior due to the initialization delay, a new data set is completed every period. The period, which corresponds to the inverse of the throughput, is a key performance-related objective for streaming applications [14, 5]. Formally, the period is the time interval between the arrival of two consecutive data sets in the application, in steady state. Given a mapping of the application onto a platform, the time spent in each resource (processor or communication link) should not exceed the period.

Finally, the target platform for this study is a chip multiprocessor (CMP), composed of $p \times q$ homogeneous cores arranged along a 2D grid. During the last century, advances in integrated circuit technology have led chip designers to increase microprocessor performance by increasing the integration density thus allowing for higher clock rates and new innovations in micro-architectures. Such innovations included wider instructions, speculative execution, branch prediction and dynamic scheduling. However, in 1996, Olukotum et al. [9] argued that such a trend would not continue because of the diminishing return caused by limited instruction level parallelism and they showed that a better way for using the denser integration would be to layout multiple simpler processors on the same chip. Moreover, power consumption consideration prevented the push towards faster clocks, thus leaving the design of chip multiprocessors as the only alternative for increasing the on-chip com-

putational capability. Specifically, increasing the number of cores rather than the processor's complexity translates into slower growth in power consumption. Currently, chip multiprocessors are commercially available and the trend is towards the continuous increase in the number of cores on single chips. The challenge is now to be able to efficiently utilize the parallelism available on chip [2].

An essential step for exploring the parallelism available in a streaming application is to provide algorithms and scheduling strategies for mapping a series-parallel graph onto a CMP, with the objective of minimizing the energy consumption while not exceeding a prescribed period. In some applications, data sets arrive at fixed time intervals, and hence the period of the application is given a priori, before any mapping is computed. In other applications, there is the freedom to choose between a set of possible periods, which are prescribed by the user. In all cases, the main goal is to reduce the energy consumption of the mapping, while enforcing the constraint on the period bound. The main contribution of this paper is the design and evaluation of a set of heuristics to solve this difficult optimization problem, building upon the theoretical results of [1]. After recalling the framework and complexity results in Section 2, we design heuristics to solve the most general problem (Section 3), and we assess their performance through simulations (Section 4). Finally, we conclude and discuss future research directions in Section 5.

2. Framework

Applicative framework. The application that is to be scheduled is a streaming application: it operates on a collection of data sets that are executed in a pipelined fashion. In this study, the application is a series-parallel graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, or SPG. Nodes of the graph correspond to different application stages, and are denoted by S_i, with $1 \leq i \leq n$, where $n = |\mathcal{S}|$ is the size of the graph. For each precedence constraint in the application, say from stage S_i to stage S_j, we have an edge $L_{i,j} \in \mathcal{E}$. For $1 \leq i \leq n$, w_i is the computation requirement of stage S_i, and for each $L_{i,j} \in \mathcal{E}$, with $1 \leq i, j \leq n$, $\delta_{i,j}$ is the volume of communication to be sent from S_i to S_j before S_j can start its computation.

An SPG is built from a sequence of compositions (parallel or series) of smaller-size SPGs. The smallest SPG consists of two nodes connected by an edge. The first node is the source of the SPG while the second is its sink. When composing two SPGs in series, we merge the sink of the first SPG with the source of the second. For a parallel composition, the two sources are merged, as well as the two sinks. We are given a *maximum elevation* of the SPG, $y_{\max}$, which is the maximum number of concurrent parallel compositions, i.e., it denotes the maximal degree of parallelism of the SPG.

Platform. The target platform is a chip multiprocessor (CMP), composed of $p \times q$ homogeneous cores $\mathcal{C}_{u,v}$, with $1 \leq u \leq p$, $1 \leq v \leq q$, arranged along a rectangular grid. There is a vertical (internal and bi-directional) communication link between $\mathcal{C}_{u,v}$ and $\mathcal{C}_{u+1,v}$, for $1 \leq u \leq p-1$, $1 \leq v \leq q$, and a horizontal link between $\mathcal{C}_{u,v}$ and

$\mathcal{C}_{u,v+1}$, for $1 \leq u \leq p$, $1 \leq v \leq q-1$. All links have the same bandwidth BW (in each direction). This means that it takes a time $\frac{\delta}{BW}$ to send δ bytes from one processor to a neighboring processor. It is possible to use only some of the communication links, and for instance to configure the $p \times q$ CMP as a $1 \times pq$ bi-directional linear array, called *bi-directional uni-line CMP*.

The voltage and frequency of each core of the CMP can be set to different values. Altogether, there is a set of possible supply voltages, together with a set of possible frequencies (or modes, or speeds), for each core. Let $\mathsf{S} = \{s^{(1)}, \ldots, s^{(m)}\}$ denote the set of all possible speeds. It takes a time $\frac{w_i}{s^{(k)}}$ to execute one data set for stage S_i at speed $s^{(k)} \in \mathsf{S}$ on a given core. Each speed induces a different dynamic power consumption, as developed below.

Mapping strategies. We discuss several mapping rules to map the SPG application onto the CMP. As for the application graph, we use *DAG-partition* mappings, which represent a trade-off between *one-to-one* and *general* mappings. The rationale is the following. One-to-one mappings obey the simplest rule: each application stage is mapped onto a distinct core. While easier to optimize and implement, this rule may be unduly restrictive, and is likely to lead to high communication costs. Obviously, it also requires that $p \times q \geq n$, thereby limiting its applicability to large platforms or small applications. A natural extension is to search for DAG-partition mappings: we first partition the initial SPG into subsets, or clusters, such that the resulting graph is acyclic. Hence this mapping rule states that if two stages S_i and S_j are in the same subset of the partition, then any other stage S_k which has an incoming dependency from S_i and an outgoing dependency to S_j, must be in the same subset of the partition. Then we map the subsets of the partition onto the cores in a one-to-one fashion. Using this mapping rule, a core which is executing a subset I of stages $\{S_i\}_{i \in I}$ will perform at most one input and one output communication for each elevation value $\{y_i\}_{i \in I}$. This is well in accordance with our initial assumption that the SPG has bounded elevation $y_{\max}$, because it ensures that each core has at most $y_{\max}$ communications to perform at each period. In contrast, a fully general mapping, that allow for arbitrary partitions of the original application graph, would require an arbitrary number of communications, only bounded by the total number of stages n, hence an unlimited amount of buffer space. Moreover, even for bounded-elevation SPGs, the problem of finding the general mapping which minimizes the energy given a period bound is trivially NP-complete (linear chain onto two processors, reduction from 2-PARTITION [6]).

Formally, the mapping is defined by an allocation function $alloc : \{1, \ldots, n\} \to \{1, \ldots, p\} \times \{1, \ldots, q\}$, which maps stages onto cores. In other words, stage S_i is mapped onto core $\mathcal{C}_{u,v}$ if and only if we have $alloc(i) = (u, v)$. Once application stages are mapped onto cores, there remains to decide how to route communications between two cores which need to communicate because of the stage assignment. Therefore, for each application edge $L_{i,j} \in \mathcal{E}$, if $alloc(i) \neq alloc(j)$, we define $path_{i,j}$ as the set of communication links that are used to communicate from core $\mathcal{C}_{alloc(i)}$

to core $\mathcal{C}_{alloc(j)}$. Note that these paths must be defined for the mapping to be fully determined.

Optimization criteria: period and energy. As motivated above, we assume that data sets arrive at regular time intervals, which is called the *period* of the application, and denoted by T. Then, given a mapping and an execution speed for each core, we can check whether the application can be executed at the prescribed rate: we must ensure that the cycle-time of each resource (computation or communication link) does not exceed T. Let $w_{u,v} = \sum_{1 \leq i \leq n | alloc(i)=(u,v)} w_i$ be the total amount of work assigned to core $\mathcal{C}_{u,v}$, running at speed $s_{u,v} \in \mathsf{S}$. The cycle-time of $\mathcal{C}_{u,v}$ for computations is $w_{u,v}/s_{u,v}$. For communications, $b_{(u,v)\to(u',v')}$, which is equal to $\sum_{1 \leq i,j \leq n | (u,v)\to(u',v') \in path_{i,j}} \delta_{i,j}$ is the number of bits sent from $\mathcal{C}_{u,v}$ to a neighbor core $\mathcal{C}_{u',v'}$[a]. The cycle-time of the communication link $(u,v) \to (u',v')$ is $b_{(u,v)\to(u',v')}/BW$. We can then compute the maximum cycle-time, which is the maximum cycle-time of all resources, and check that it is not greater than T.

Once an SPG application has been mapped onto the CMP, there are two sources of *energy consumption*: the cores consume energy for computations and the routers consume additional energy for communications.

For the computations, we assume that each core involved in the execution consumes some static energy during the whole period T, and some dynamic energy that depends on the amount of operations, and on the speed at which these operations are executed. Let $\mathcal{A}$ be the set of active cores: $\mathcal{A} = \{\mathcal{C}_{u,v},\ 1 \leq u \leq p, 1 \leq v \leq q \mid \exists\ 1 \leq i \leq n,\ alloc(i) = (u,v)\}$. The total energy consumed for computations is $E^{(\text{comp})} = |\mathcal{A}| \times P^{(\text{comp})}_{\text{leak}} \times T + \sum_{\mathcal{C}_{u,v} \in \mathcal{A}} \frac{w_{u,v}}{s_{u,v}} \times P^{(\text{comp})}_{s_{u,v}}$, where T is the prescribed period, $P^{(\text{comp})}_{\text{leak}}$ is the leakage power dissipated together with computations, and $P^{(\text{comp})}_{s_{u,v}}$ is the dynamic power associated with speed $s_{u,v}$.

For the communications, there is also a static part due to leakage, which is paid for all cores: even if a core is not enrolled in the computation, its routers and communication links may be used to route data between remote processors. The dynamic part is directly proportional to the number of bits that are sent across each link. Hence, $E^{(\text{comm})} = P^{(\text{comm})}_{\text{leak}} \times T + \left(\sum_{u,v} \sum_{u',v'} b_{(u,v)\to(u',v')}\right) \times E^{(\text{bit})}$, where T is the period, $P^{(\text{comm})}_{\text{leak}}$ is the aggregated leakage power dissipated by all routers and links, and $E^{(\text{bit})}$ is the energy to transfer a bit across neighboring cores. Finally, the total energy consumption is $E = E^{(\text{comp})} + E^{(\text{comm})}$.

Optimization problem. We are ready to formally define the optimization problem: given a bounded-elevation SPG and a period threshold T, find a mapping whose maximal cycle-time does not exceed T and whose energy E is minimum.

The only polynomial instance of this problem is for the uni-directional uni-line CMP. In this case, there is a dynamic programming algorithm that finds the optimal solution [1]. It is worth noting that this polynomial instance becomes NP-complete

[a] (u'=$u+1$ and v'=v) or (u'=$u-1$ and v'=v) or (u'=u and v'=$v+1$) or (u'=u and v'=$v-1$).

for SPGs of unbounded elevation. All other problem instances are NP-hard, in particular for bi-directional 2D meshes.

3. Heuristics

Random. This heuristic calls a procedure that works in two steps. First, we randomly build a DAG-partition of the initial SPG, while ensuring that the period is matched for computations: we choose randomly a speed for the core which will handle the current subgraph G (initially, the source of the SPG), and we keep a list of stages of the SPGs that can be added to G while maintaining a DAG-partition. We pick a stage from this list randomly as long as computations do not exceed the period. When moving to the next core, we choose the first stage in the current list and iterate. In a second step, we decide randomly on which core each subgraph is mapped, and communications are done following a XY routing: a communication from $\mathcal{C}_{u,v}$ to $\mathcal{C}_{u',v'}$ follows horizontal links from $\mathcal{C}_{u,v}$ to $\mathcal{C}_{u,v'}$, and then vertical links from $\mathcal{C}_{u,v'}$ to $\mathcal{C}_{u',v'}$. If the period is not exceeded on any communication link, then the mapping is valid, otherwise there is no solution. For each problem instance, **Random** calls ten times this procedure, and keeps the solution that minimizes the energy consumption, if there is at least one valid solution; otherwise it fails. This heuristic performs a random mapping, and it is used for comparison purposes.

Greedy. This heuristic greedily assigns the SPG onto the platform, on which all cores are running at speed s. We try all possible speed values $s \in \mathsf{S}$, and keep the best solution. Given a speed $s \in \mathsf{S}$, we keep a list of cores that are ready to be processed, and for each core, a list of successors, together with the corresponding outgoing communications. Initially, the only core in the list is $\mathcal{C}_{1,1}$, and we assign to this core the source stage S_1. The corresponding list of successors corresponds to the successors of S_1 in the SPG, and they are sorted by non-increasing communication volume to S_1. When we process a core $\mathcal{C}_{u,v}$, we successively try to add some of the successors (from the current list) to this same core until the list is empty or the period is exceeded for computations on $\mathcal{C}_{u,v}$. For each set of stages mapped onto $\mathcal{C}_{u,v}$ and the corresponding list of successors, we greedily share the corresponding communications between neighboring cores $\mathcal{C}_{u,v+1}$ and $\mathcal{C}_{u+1,v}$: communications are taken from the sorted list and assigned to the core that has currently the smallest amount of incoming communications. Then, we check that the partitioning is correct (no cycles in the dependence graph, i.e., we have a DAG-partition), and we check whether the bound on the period is achieved, both for computations and communications. If it is correct, we save the current solution before adding one more stage onto core $\mathcal{C}_{u,v}$ and iterating. At the end of the iteration, we keep the last valid (saved) solution, i.e., the valid solution with the most number of stages onto $\mathcal{C}_{u,v}$. Cores $\mathcal{C}_{u,v+1}$ and $\mathcal{C}_{u+1,v}$ are then added to the list of ready cores, together with the list of successors (i.e., the stages that can either be assigned to this core, or forwarded to the neighboring cores).

The procedure finishes when the list of ready cores is empty, which means that

all stages have been processed. Otherwise, the heuristic fails, and we move to the next speed. The energy for the mapping obtained with a given speed is computed by first *downgrading* the speed of each core, if possible: the procedure returns the mapping, and then we compute the amount of computations on each core, and set the core to the slowest possible speed, in order to save energy. Cores that are not used are turned off. Finally, the **Greedy** heuristic selects the mapping which corresponds to the lowest energy consumption.

2D dynamic programming algorithm DPA2D. This heuristic starts by mapping the initial SPG onto an $x_{\max} \times y_{\max}$ grid (x_i is defined as the number of stages along a longest path from the source node to S_i, and $x_{\max} = \max_{1 \leq i \leq n} x_i$). Then, this grid is mapped onto the CMP, thanks to a double nested dynamic programming algorithm. First, we perform a dynamic programming algorithm to cut the grid into a set of columns, which are to be mapped onto a column of cores. Let $\mathcal{F}(m, v, D)$ be the optimal energy consumption to compute the first m levels of the SPG (i.e., all stages S_i with $x_i \leq m$), using v columns of cores, regardless of the outgoing communications. D is then the corresponding distribution of outgoing communications, i.e., a list of triplets (y, b, i), where y is the row from which communication is outgoing (i.e., the communication is initiated by core $\mathcal{C}_{y,v}$), b is the amount of data, and S_i is the destination stage. We enforce these communications to go through $\mathcal{C}_{y,v+1}$, and then the communication will be redistributed to the destination core through vertical links. The solution is $\mathcal{F}(x_{\max}, q, D)$, and the recurrence is written as:

$$\mathcal{F}(m, v, D) = \min_{m' < m} \left(\mathcal{F}(m', v-1, D') + \mathcal{F}^{\text{comm}}(D') + \mathcal{F}^{\text{col}}(m'+1, m, D', D) \right),$$

with the initialization $\mathcal{F}(m, 1, D) = \mathcal{F}^{\text{col}}(1, m, \emptyset, D)$.

D' is the distribution of outgoing communications corresponding to the m' which leads to the optimal energy consumption, i.e., obtained with $\mathcal{F}(m', v-1, D')$. $\mathcal{F}^{\text{comm}}(D')$ is the energy consumption induced by communications from column $v-1$ to column v (on horizontal links), given the distribution D' of outgoing communications of column $v+1$. If the bandwidth is exceeded on one of these horizontal links (i.e., $\exists 1 \leq y \leq p$ such that $\sum_{(y,b,i) \in D'} b > BW$), we set $\mathcal{F}^{\text{comm}}(D') = +\infty$. $\mathcal{F}^{\text{col}}(m_1, m_2, D', D)$ is the optimal energy consumption of the column of the CMP which is processing stages S_i with $m_1 \leq x_i \leq m_2$: it accounts both for computations, and for vertical communications in the column, given the distribution of outgoing communications of the previous column, D'. The distribution of outgoing communications of this column is then D. Note that in the recurrence, D is an output of $\mathcal{F}^{\text{col}}(m'+1, m, D', D)$, while D' is an output of $\mathcal{F}(m', v-1, D')$. The values of $\mathcal{F}^{\text{col}}$ (and therefore, distribution D) are computed thanks to another dynamic programming algorithm: we compute $\mathcal{F}^{\text{col}}_{(m_1,m_2,D',D)}(g, u)$, which corresponds to the mapping of stages S_i, with $m_1 \leq x_i \leq m_2$ and $y_i \leq g$, onto the u first cores of a column of the CMP. As before, D' is an input, it corresponds to the distribution of outgoing communications arriving into the current column, while D is the distribution of out-

going communications of the current column for the solution which minimizes the energy consumption. Then we have $\mathcal{F}^{\text{col}}(m_1, m_2, D', D) = \mathcal{F}^{\text{col}}_{(m_1,m_2,D',D)}(y_{\max}, p)$.

For the distribution within a column, the recurrence writes:

$$\mathcal{F}^{\text{col}}_{(m_1,m_2,D',D)}(g,u) = \min_{g' \leq g} \Big(\mathcal{F}^{\text{col}}_{(m_1,m_2,D',D)}(g', u-1) + \mathcal{F}^{\text{cal}}_{(m_1,m_2,D)}(g'+1, g) + \mathcal{F}^{\text{ver}}_{(m_1,m_2,D')}(g'+1, g, u-1) \Big),$$

with the initialization $\mathcal{F}^{\text{col}}_{(m_1,m_2,D',D)}(0,u) = 0$, and no outgoing communications from row 1 to row u, except the communications from D' that must be forwarded to the next column.

$\mathcal{F}^{\text{ver}}_{(m_1,m_2,D')}(g'+1, g, u-1)$ is the energy consumption of the vertical communications between cores $u-1$ and u in the column. These communications can either come from two dependent stages of the column, or be forwarded from the previous column (D'). If the bandwidth of the link is exceeded, we set the value to $+\infty$. Finally, $\mathcal{F}^{\text{cal}}_{(m_1,m_2,D)}(g'+1, g)$ is the optimal energy consumption of a core which is computing all stages S_i such that $m_1 \leq x_i \leq m_2$, and $g'+1 \leq y_i \leq g$. If the period constraint cannot be fulfilled, or if the corresponding partition does not fulfill the DAG-partition constraint, the value is set to $+\infty$. Moreover, this function is adding to distribution D the communications from a stage S_i to another stage S_j, with $x_j > m_2$. These communications will occur on row u. Note that in the recursive computation of $\mathcal{F}^{\text{col}}$, we can have $g' = g$, which means that no stage is assigned to core $\mathcal{C}_{u,v}$. This may happen if there are not enough stages in the column, or if this would save communications.

1D heuristics. The two last heuristics configure the CMP as a uni-directional uni-line CMP with $r = p \times q$ cores, by embedding it into the bi-directional platform as a *snake*:

$$\begin{array}{l} \mathcal{C}_{1,1} \rightarrow \mathcal{C}_{1,2} \rightarrow \ldots \rightarrow \quad \mathcal{C}_{1,q} \\ \qquad\qquad\qquad\qquad\qquad\quad \downarrow \\ \mathcal{C}_{2,1} \leftarrow \ldots \leftarrow \mathcal{C}_{2,q-1} \leftarrow \mathcal{C}_{2,q} \\ \downarrow \\ \mathcal{C}_{3,1} \rightarrow \mathcal{C}_{3,2} \rightarrow \ldots \end{array}$$

The **DPA1D** heuristic computes the optimal solution of the dynamic programming algorithm of [1] with $r = p \times q$ cores. The mapping is then done along the snake; no other communication link is used. Note that if the SPG is a linear chain, even if there are communication costs, then this heuristic is optimal, since any other solution could not exploit the communication links discarded with the snake structure. It is also optimal for any SPG without communication. However, **DPA1D** may take wrong decisions when communications are intensive, since it is restricted to a subset of communication links. Moreover, its complexity of $O(p \times q \times n \times n^{y_{\max}})$ makes it intractable for SPGs with large $y_{\max}$.

Finally, the **DPA2D1D** heuristic computes the solution with the **DPA2D** heuristic on a $1 \times r$ CMP, and then does the mapping along the snake, similarly to

DPA1D. The goal of this heuristic is to obtain efficient solutions when communications are not too intensive, and when the optimal **DPA1D** cannot find a solution in reasonable time.

4. Simulation results

This section reports simulation results assessing the performance of the various heuristics. As for the applications, we use both real-life applications taken from the *StreamIt* suite [13], and randomly generated applications, which allows us to cover a broader spectrum. As for the target platform, we use 4×4 and 6×6 CMP grids, whose hardware characteristics are representative of state-of-the-art devices. The source code for all simulations is publicly available at [11].

4.1. *Simulation setting*

***StreamIt* suite.** There are 12 workflows in the *StreamIt* suite [13]. Their main characteristics are summarized in Table 1, where we give the size n, the maximum label values $y_{\max}$ and $x_{\max}$, and their *computation-to-communication ratio (CCR)*, defined as the sum $\sum_{i=1}^{n} w_i$ of all computations over the sum $\sum_{L_{i,j} \in \mathcal{E}} \delta_{i,j}$ of all communications. We observe in Table 1 that all workflows have a large CCR, hence are compute-intensive rather than data-intensive. In the simulations, we first use the workflows as such, with the original CCR values, and then we scale communication weights (the $\delta_{i,j}$) to change each CCR successively to 10, 1, and 0.1, so as to assess the impact of the communications on the performance of the heuristics.

Randomly generated applications. We randomly build SPG applications (by applying recursively series and parallel compositions of SPG applications), and we extract their size n, their elevation $y_{\max}$, together with their computation-to-communication ratio (CCR).

Index	Name	n	$y_{\max}$	$x_{\max}$	CCR
1	Beamformer	57	12	12	537
2	ChannelVocoder	55	17	8	453
3	Filterbank	85	16	14	535
4	FMRadio	43	12	12	330
5	Vocoder	114	17	32	38
6	BitonicSort	40	4	23	6
7	DCT	8	1	8	68
8	DES	53	3	45	7
9	FFT	17	1	17	17
10	MPEG2-noparser	23	5	18	9
11	Serpent	120	2	111	9
12	TDE	29	1	29	12

CMP configuration. For processor speeds and power consumption, we use the model of [4, 8], with five speeds for each core $s_{u,v} = (0.15, 0.4, 0.6, 0.8, 1)\ GHz$, and corresponding power consumptions $P_{s_{u,v}}^{(\text{comp})} = (80, 170, 400, 900, 1600)\ mW$. The power consumption of the processor when it is idle is $P_{\text{leak}}^{(\text{comp})} = 80mW$. We use 16-byte wide communication links [10], whose bandwidths are $BW = 16 \times 1.2\ Gbytes$, which is reasonable according to [10]. Note that from the communication perspective, decreasing CCR has the same effect on the results as decreasing the width of the communication link below 16 bytes. The link energy is assumed to be between 1 and 10 picojoule per bit [3]; we fix $E^{(\text{bit})} = 6pJ$. Finally, we use $P_{\text{leak}}^{(\text{comm})} = 0$ without loss of generality (because for all heuristics the same quantity $P_{\text{leak}}^{(\text{comm})} \times T$ will be added to the total energy).

Period bound T. We need to find a meaningful value of T for each workflow. Indeed, if T is too large, all heuristics will map all stages onto a single processor running at the slowest speed, while if T is too small, all heuristics will fail. We choose T as follows: for each workflow, we start with $T = 1s$. With such a period, we observe that at least one heuristic succeeds. Then we iteratively divide the period by a factor of 10 and run all heuristics under this new value until all heuristics fail. We retain the period as the penultimate value, which is the last one before total failure. Note that this value depends upon the workflow, and that it is chosen to give some tightness to the mapping problem: at least one heuristic succeeds to find a mapping that matches the bound T, but none does for $T/10$.

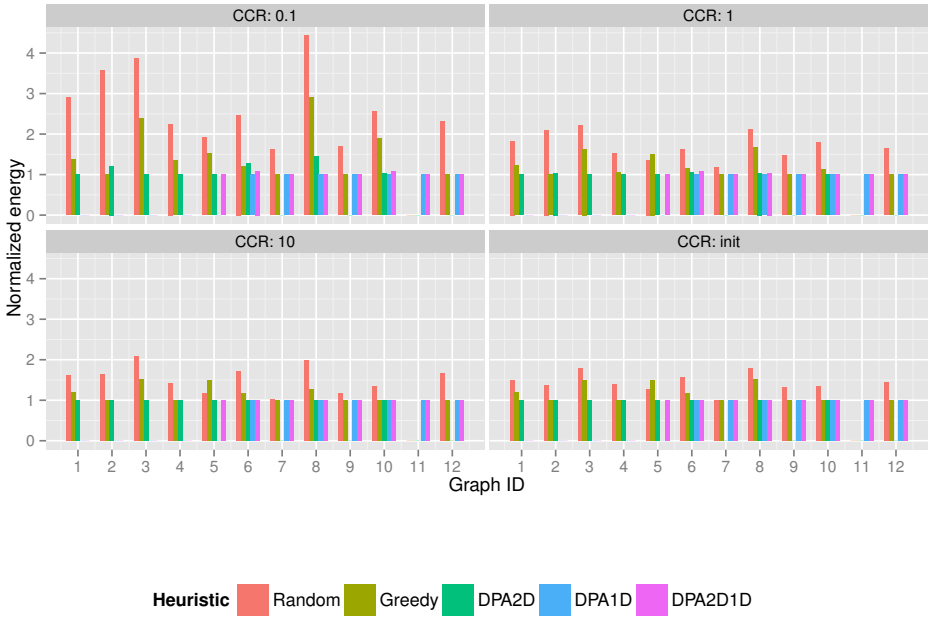
4.2. *Simulation results*

4.2.1. StreamIt *suite*

In Figure 1, we plot the energy computed by the five heuristics for each application, given a CMP size (4×4 or 6×6) and a CCR ratio (set to the original value, 10, 1 and 0.1). On the horizontal x axis, each group corresponds to an application, and x is the number of the application in Table 1. On the vertical axis, we plot the energy found by each heuristic, normalized by the minimum value obtained over all heuristics (so that the best heuristic returns 1, and the other ones return higher values). The **DPA1D** heuristic fails to return a solution for the first four applications, because there are too many possible splits to explore, and it is not plotted for those applications. More generally, each time a heuristic fails on a given application, it does not appear on the corresponding graph.

4×4 CMP grid. Results for a 4×4 CMP grid are given in Figure 1(a). When computations are predominant, i.e., when the CCR is set to its original value, or uniformly equal to 10, we observe that **Greedy**, **DPA2D**, **DPA1D** and **DPA2D1D** return similar results, and that **Random** always is within a factor of two. We also observe that **DPA2D** often fails on graphs with small elevation (linear graphs), because it wastes a lot of cores. For instance, if the application is exactly a pipeline

(a) CMP 4×4



(b) CMP 6×6

Figure 1. Normalized energy on the set of *StreamIt* applications.

(workflows numbered 7, 9 and 12), **DPA2D** can only enroll 4 cores over the 16 that are available. This fact holds true irrespective of the CCR.

When communications are more important, i.e., when the CCR is uniformly set to 1 or 0.1, **Random** gets much worse than the other heuristics: if it does not fail, its energy is between 2 and 4 times worse than the best one. In a general manner, we see that **DPA2D** is the best heuristic when the application graph has a high elevation. We point out that **DPA1D** and **DPA2D1D** are the only successful heuristics for the workflow 11, whatever the CCR ratio is. This workflow fits very well with the main design idea of **DPA1D** and **DPA2D1D**: it is a pipeline-like graph (its elevation is only 2) with numerous stages. The other heuristics fail to find a good load-balance of computations and communications for this application. The difference between **DPA1D** and **DPA2D1D** is tiny: when **DPA1D** finds a solution, **DPA2D1D** finds a close one, and there is only one graph (numbered 5) on which **DPA2D1D** succeeds, whereas **DPA1D** fails, because of the high memory complexity. Note that, in some cases, the solution of **DPA1D** is better than that of **DPA2D1D**, confirming that **DPA2D1D** does not return the optimal 1D mapping.

Altogether, **Greedy** seems to be a general-purpose heuristic that succeeds on most graphs, and it is always superior to **Random**. On the contrary, **DPA1D**, **DPA2D1D** and **DPA2D** are "specialized" heuristics, the first two heuristics are very efficient for long and almost linear graphs but not good for fat graphs of large elevation, and the last one behaving just as the opposite.

6×6 CMP grid. Results for a 6×6 CMP grid are given in Figure 1(b). Because the target grid is larger, it is easier to find a mapping that matches the period bound, especially for applications with a small number of stages. This is quantified in Table 2, where we report the number of failures for each heuristic.

We observe that the difference between solutions of **DPA2D1D** and solutions of **DPA1D** almost disappears. Otherwise, the conclusion remains more or less the same as on the 4×4 CMP grid, with **Greedy** always successful but also always inferior to one of the three specialized heuristics, **DPA1D**, **DPA2D1D** and **DPA2D**, depending upon the graph shape.

Platform size	**Random**	**Greedy**	**DPA2D**	**DPA1D**	**DPA2D1D**
4×4	5	4	16	20	16
6×6	0	0	17	20	8

4.2.2. *Random SPGs*

For the randomly generated SPGs, we plot six groups of two graphs; in each group, the two graphs correspond to a number of nodes n that is either 50 or 150, and are obtained for a given CCR (10, 1 or 0.1) and for a given number of cores p in a row of the square CMP (4 or 6). On the horizontal axis, we represent the elevation of

CCR	**Random**	**Greedy**	**DPA2D**	**DPA1D**	**DPA2D1D**
10	58	56	156	1516	2
1	58	56	156	1520	4
0.1	300	287	348	1340	916

the SPG. For each value of the elevation, we average the results obtained on 100 randomly generated applications. On the vertical axis, we plot the inverse of the energy found by each heuristic, normalized to the minimum value obtained over all heuristics (so that the best heuristic returns 1, and the other ones return smaller values).

With 50 nodes and a 4×4 CMP grid. Results are given in Figure 2. When computations are predominant, i.e., when the CCR is uniformly equal to 10, we observe that the two 1D heuristics always return good results. For small elevations, **DPA1D** is the best, but it often fails as soon as the elevation is greater than 4, thus leading to poor results. **DPA2D1D** returns very good results whatever the elevation of the graph. The 2D heuristic **DPA2D** is the best for elevations greater than 6, but it often fails on graphs with small elevation, because it wastes a lot of cores. For instance, if the application is exactly a pipeline (elevation 1), **DPA2D** can only enroll 4 cores over the 16 that are available. This fact holds true irrespective of the CCR. **Greedy** and **Random** are not as good, but **Greedy** always outperforms **Random**.

When communications and computations are more balanced (CCR of 1), similar results can be observed, but **DPA2D1D** is a bit further from the best solution, since it cannot utilize all the communication links. Finally, for communication-intensive applications (CCR of 0.1), **Random** gets much worse than the other heuristics: its energy can be up to 10 times worse than the best one. Also, the 1D heuristics do not perform well, except for small elevation graphs, because of their restriction in the communication pattern. In a general manner, we see that **DPA2D** is the best heuristic when the application graph has a high elevation.

Number of failures. In Table 3, we report the number of failures for each heuristic, again with 50 nodes and a 4×4 CMP grid. With a large CCR (10 or 1), **DPA2D1D** almost always succeeds to find a solution, which are in turn pretty good (see Figure 2). **Greedy** is always reasonably robust, whatever the CCR, and is followed closely by **Random**. **DPA2D** fails a bit more frequently because it does not often succeed with graphs of small elevation, as explained earlier. Finally, **DPA1D** succeeds only for graphs of small elevation, which leads to a very high failure rate.

Other results. We have performed further simulations on larger applications and/or different CMP grid sizes, see Figures 2 and 3. Overall, the conclusions remain the same, and they confirm the results derived from the real-life *StreamIt* applications.

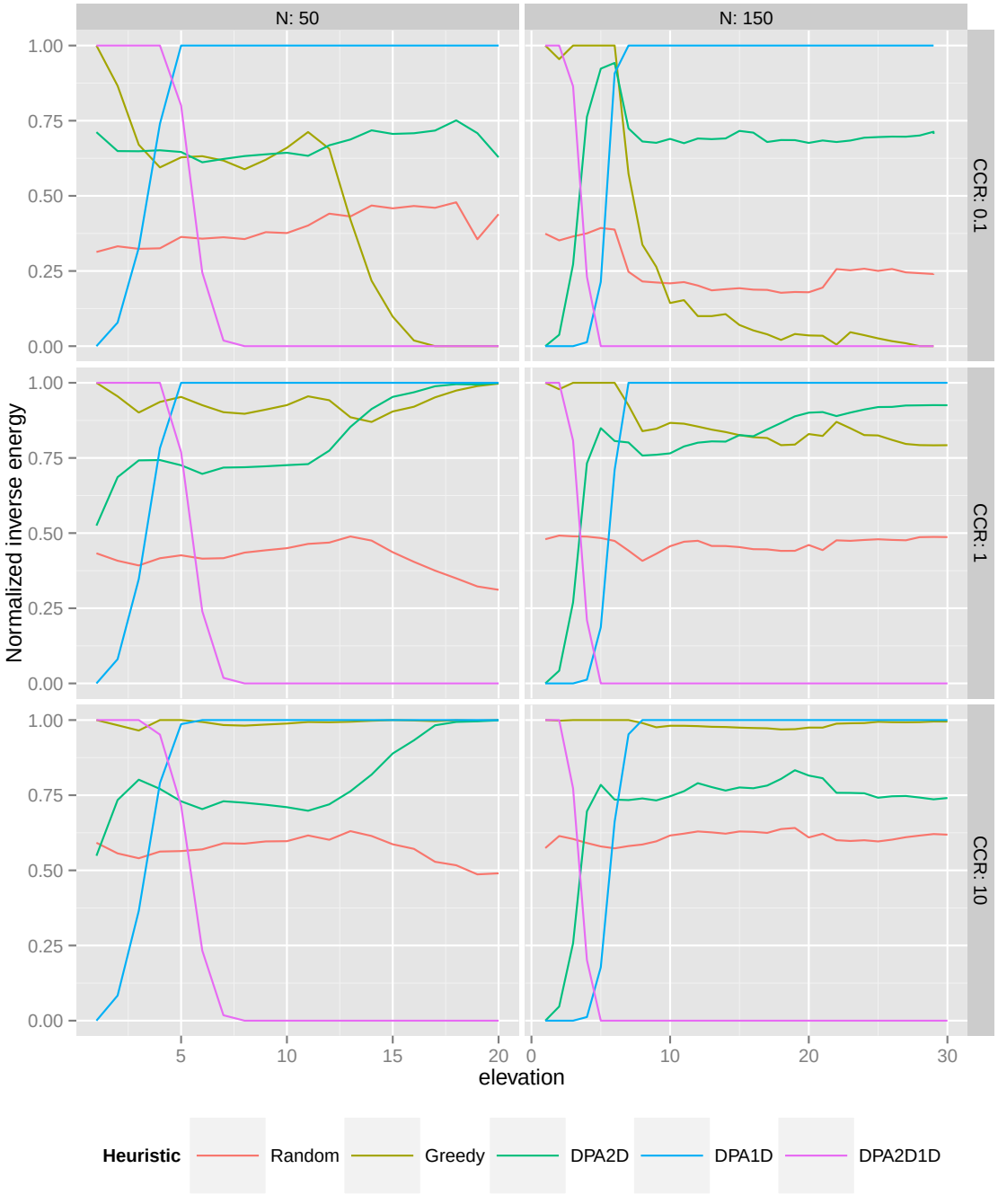


Figure 2. Normalized energy inverse on a random set of applications for a 4×4 CMP grid.

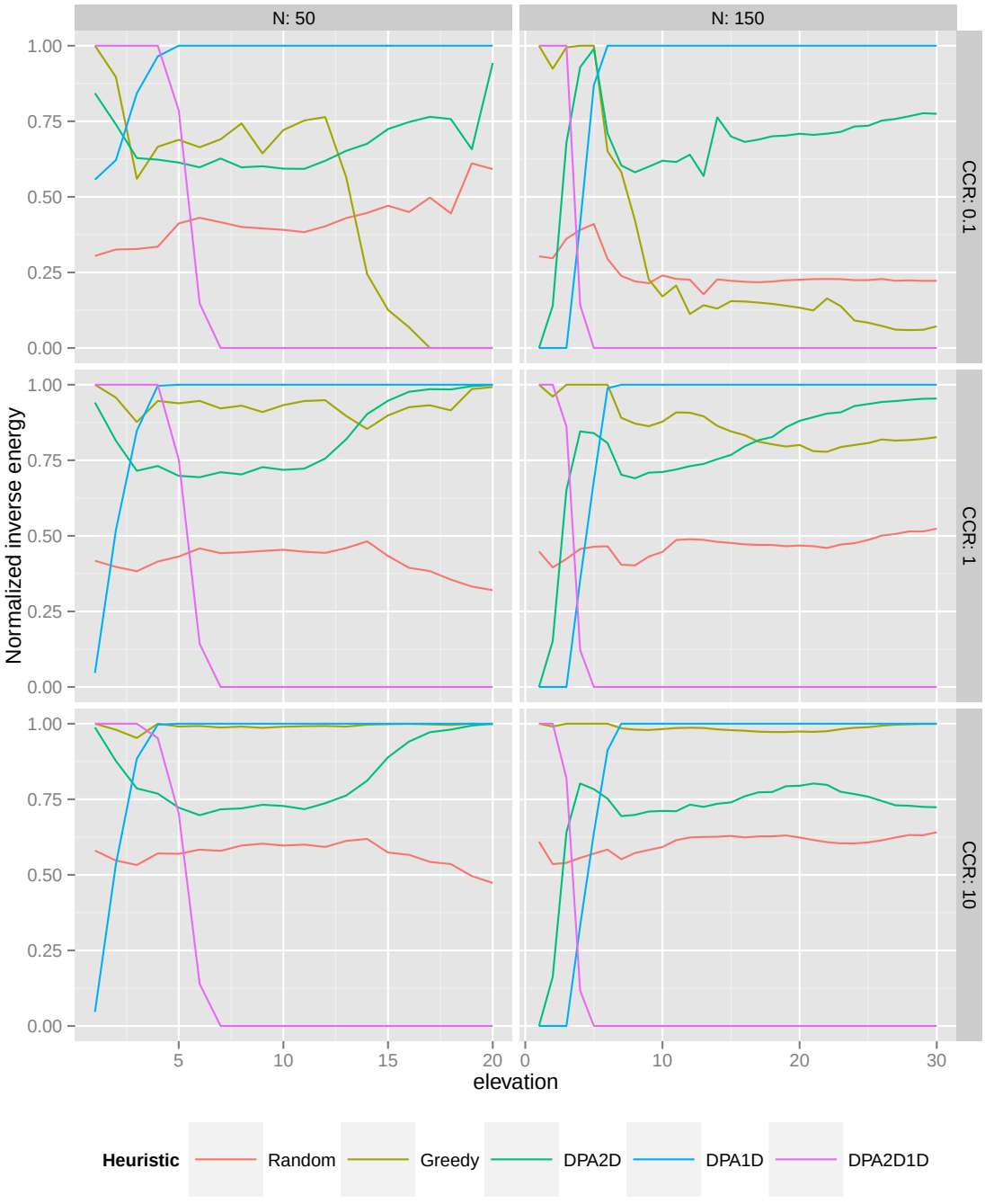


Figure 3. Normalized energy inverse on a random set of applications for a 6×6 CMP grid.

5. Conclusion

This paper contributes to the efficient utilization of multicores by considering an important class of streaming applications that can be modeled by a series-parallel graph, and studying the problem of mapping these applications to 2-dimensional tiled CMP architectures. The objective of the mapping is to minimize the energy consumption while maintaining a given level of performance, reflected by the processing rate of the data streams. While both processing and communication capabilities are considered during the mapping, only the processing power can be managed through dynamic voltage and frequency scaling.

This is a first attempt to propose practical solutions to the problem, and to the best of our knowledge, there are no other heuristics from the literature to give a basis for comparison, and we rather compare our heuristics to a random mapping. The simulations conducted with the *StreamIt* suite and the randomly generated SPGs confirm the efficiency of the main design principles underlying the various heuristics. While **Greedy** is the most robust approach, it is always superseded by at least one of the three specialized algorithms, **DPA1D** for long pipeline-like graphs, **DPA2D** for fat graphs of large elevation or **DPA2D1D** for any graph containing low communication weights and for graphs of low elevation. While there is no absolute winner, in practice the shape of the task graph is given, and the best heuristic can be selected accordingly. Furthermore, one could run several heuristics and select the best result. Indeed, all heuristics have low complexity, so that (i) their execution time will very likely be negligible in front of the application execution time; and (ii) they are expected to scale well for larger CMP architectures (of size 16×16 or even larger).

Future research may investigate general mappings, and assess the difference with DAG-partition mappings, both from a theoretical and a practical perspective. It would also be interesting to consider systems in which the communication power can also be managed.

Acknowledgments

We would like to thank the reviewers, whose comments and suggestions greatly helped us to improve the final version of the paper. This work has been supported in part by the ANR RESCUE project. Anne Benoit and Yves Robert are with the Institut Universitaire de France.

References

[1] A. Benoit, R. Melhem, P. Renaud-Goud, and Y. Robert. Energy-aware mappings of series-parallel workflows onto chip multiprocessors. Research Report 7521, INRIA, France, Jan. 2011. Available at `http://graal.ens-lyon.fr/~abenoit/`. Short version appeared in ICPP'2011.

[2] G. Blake, R. Dreslinski, and T. Mudge. A survey of multicore processors. *Signal Processing Magazine*, 26(6):26–37, Nov. 2009.

[3] G. Chen, F. Li, M. Kandemir, and M. J. Irwin. Reducing NoC energy consumption through compiler-directed channel voltage scaling. *SIGPLAN Not.*, 41:193–203, June 2006.

[4] J.-J. Chen and T.-W. Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Proc. of ICCAD'07, the Int. Conf. on Computer-Aided Design*, pages 289–294, 2007.

[5] DataCutter. DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment. `http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm`, -.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[7] M. P. Mills. The internet begins with coal. *Environment and Climate News*, 1999.

[8] L. Niu. Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee. In *Proc. of RTCSA, the 16th Int. Conf. on Embedded and Real-Time Comp. Syst. and App.*, pages 163 –172, 2010.

[9] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. *SIGPLAN Not.*, 31:2–11, Sept. 1996.

[10] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research Challenges for On-Chip Interconnection Networks. *IEEE Micro*, 27:96–108, Sept. 2007.

[11] P. Renaud-Goud. Source Code for the Experiments, 2011. `http://graal.ens-lyon.fr/~prenaud/sp-cmp/`.

[12] F. Schueller, J. Qin, F. Nadeem, R. Prodan, T. Fahringer, and G. Mayr. Performance, Scalability and Quality of the Meteorological Grid Workflow MeteoAG. In *Proc. of 2nd Austrian Grid Symp.*, 2006.

[13] StreamIt Project. `http://groups.csail.mit.edu/cag/streamit/apps/stream-graphs`, 2008.

[14] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Proc. of PPoPP'95, the 5th Symp. on Principles and Practice of Parallel Programming*, 1995.